Open-Endedness of Objects and Types in Martin-Lof's Type Theory

AUTHOR(S):

Tsukada, Yasuyuki

CITATION:

Tsukada, Yasuyuki. Open-Endedness of Objects and Types in Martin-Lof's Type Theory. 数理解析研究所講究録 1993, 851: 102-126

ISSUE DATE:

1993-10

URL:

http://hdl.handle.net/2433/83701

数理解析研究所講究録
第 851 巻 1993 年 102-126

Open-Endedness of Objects and Types in Martin-Löf's Type Theory

Yasuyuki Tsukada
NTT Basic Research Laboratories
3-9-11 Midori-cho, Musashino, Tokyo 180, Japan
tsukada@ntt-20.ntt.jp

Abstract

This paper presents a comprehensive formulation of *open-endedness* of types as well as objects in Martin-Löf's type theory. This formulation is a natural generalization of Allen's non-type-theoretical reinterpretation of the theory, and demonstrates a structural extension of Howe's formulation of computational open-endedness. Suppose that a language underlying the theory is specified as a *method system*, which consists of a *preobject system* as the computational part and a *pretype system* as the structural part. Then types and their objects are uniformly and inductively constructed as a *type system* that is built from the method system and that can provide a semantics of the theory. The main theorem shows that the original inference rules concerning objects or types remain valid in any type system built from a *deterministic* and *regular* extension of the original method system. This result includes a prescription for the class of types that can be introduced into the theory, which prescription is useful for checking whether specific new types can be introduced.

1 Introduction

This paper provides a mathematical interpretation of the "open-endedness" property in Martin-Löf's type theory. This semantical property, which has played a vital role in the design of the theory, is here formulated and studied from the mathematical viewpoint. We begin with an informal and general definition of open-endedness.

1.1 Informal Definition of Open-Endedness

In general, a *theory*—that is, a framework composed of languages, their semantics, and inference rules—is called *open-ended* provided that,

> when a programmer or a mathematician using the theory works in a language L_0 whose semantics $M(L_0)$ is given by a mapping M, any possible sequence $L_0 \sqsubseteq L_1 \sqsubseteq L_2 \sqsubseteq \cdots$ of extended languages satisfies the following conditions:[1]
>
> - *semantical anticipation*: M also gives the semantics $M(L_i)$ of L_i for every $i \geq 1$ "effectively";
> - *validity persistency*: every inference rule in L_0 is valid in $M(L_i)$ for all $i \geq 0$.

In other words, an open-ended theory is one whose inference rules remain valid under extensions of its underlying language.

Extensibility of language is one of the most important requirements for a basic theory for "interactive" proof development systems, which need to support the dynamic and partial reasoning processes of human thought. It should be possible for newly invented concepts, such as novel proof techniques and original data types, to be introduced into the current language. Instead of simply "adding" them from outside the language, we can of course adhere to "defining" them within the language. However, the flexibility inherent in being able to add them is more important than the stability resulting from being able to introduce them only by definitions. This is because such definitions, even though they are possible, often force us to handle very delicate and complicated encodings. Each concept should be introduced in its most appropriate form, so we should consider a theory that has the open-endedness property.

Martin-Löf's Intuitionistic Type Theory (ITT) [14, 15], the basis for such interactive proof development systems as those described in [8, 18], is open-ended and designed to anticipate the introduction of new objects and types. Open-endedness in ITT can be viewed generally as follows.

1.2 Open-Endedness in ITT

The open-endedness property comes from *self-containedness*, which is essential to any fundamental theory for constructive mathematics. In general, we can specify the semantics of a formal language used in programming or mathematics by means of modeling, or synonymously, translation into another language. So, how many times should we repeat this process? We would like to obtain an ultimate theory whose languages have no need of further modelings or translations. It is natural that this theory should based on the most basic concepts explained only by purely semantical means. As a result of semantical investigations, ITT was proposed as an example of such an ultimate theory [16].

The selected basic concepts in ITT, *objects* and *types*, are explained by purely semantical means in terms of the "primitive" notion, *methods*.[2] In other words, those concepts on which ITT is built are naturally "open." This inherent *open-endedness of objects and types* is the reason why ITT anticipates the introduction of new objects and types. For example, a judgement of the form "A type," which says that A is a *type*, means that the method A has a *canonical*

[1] These conditions are not trivial because this paper will consider "semantics" to be "term models."

[2] A similar situation can be found in the Brouwer-Heyting-Kolmogorov (BHK-) interpretation of the fundamental logical connectives; indeed, *proofs* of the *propositions* built from such connectives are semantically explained in terms of the primitive notion, *methods*. Consequently, *open-endedness of proofs* follows.

type as its value, where the type denoted by A is defined by prescribing how a *canonical object* of the type is formed as well as how two *equal canonical objects* of the type are formed. There is no limitation on this prescription except that the equality relation in a canonical type must be reflexive, symmetric, and transitive. The definition of a canonical type is thus fairly arbitrary, and this results in open-endedness of types. Another kind of judgement, of the form "$a \in A$," presupposes that A is a type and asserts that a is an *object* of type A. This means that the method a has as its value a canonical object of the canonical type denoted by A. There is no intensional restriction on the method a in this explanation, and this lack of restriction produces open-endedness of objects.

This type-theoretical semantics is useful for a basic theory for interactive proof development systems. For example, suppose that a programmer or a mathematician wants to define the type N of natural numbers. The conventional definition, which says that N consists of $0, \mathrm{S}(0), \mathrm{S}(\mathrm{S}(0)), \ldots$, is not sufficient because possible future members such as 10^{10}, $10 + 10$, 10000!, and "a natural number expressed by using some real-valued functions introduced later" are not considered when N is defined. This problem disappears in the type-theoretical semantics because open-endedness of objects guarantees that the definition of N anticipates the introduction of new natural numbers. Open-endedness of objects also supports such mechanisms as (1) an object of type $A \rightarrow B$ is applicable to every possible future member of A, and (2) a family $D(x)$ of types over C is defined in such a way as to anticipate the introduction of new objects of C. Furthermore, by virtue of open-endedness of types, a universe of types is defined so that new, previously unimagined types may later be introduced into the universe. This feature of ITT is expected to overcome the difficulties of reflecting the dynamism of human thought, whereas conventional static theories are restricted by the necessity of describing their entire structure in advance in rigorous detail. There is a need, however, to bring this profound but rather philosophical open-endedness into sharper relief so that it may be used to design new systems for programming and mathematics.

1.3 Overview of This Paper

This paper presents a comprehensive formulation of open-endedness of objects and types in ITT; that is, a complete demonstration of the semantical anticipation and validity persistency conditions with regard to ITT. This requires the formulation of the following notions: (**a**) *language*, (**b**) *extension*, and (**c**) *semantics*.

a) The class of *method systems* is introduced to define an open-ended body of languages underlying the theory. A method system consists of two parts. One—the "computational" part—is a *preobject system*, which is the same as a *structured lazy computation system* in [12, 13, 6]. It specifies a set of *operators*, a set of *canonical* operators, and a set of *evaluation rules*. Then the set of *terms*, the set of *canonical* terms, and the *lazy evaluation relation* between two closed terms, are generated. The other—the "structural" part—is a *pretype system* defined on the preobject system. It grants to special canonical operators the privileges of *type constructors*, and it assigns to each of the type constructors the following:

- the *kind* of the type constructor—any of the type constructors in ITT can be regarded as a map that assigns a new canonical type to bundles of "families of types" and associated "functions," and the notion of kind is abstracted from the domains of such maps;

- an expression that can serve as a *strictly positive inductive definition* of the equality relation in a canonical type formed by the type constructor—the class of such expressions is systematically defined by using the kind of the type constructor. (Details are presented in Section 2.)

b) The notion of *extension* of a method system is defined in such a way as to preserve the meaning of the original system. (Details are presented in Section 3.)

c) Suppose that a language underlying the theory is specified as a method system L. Then *types* and their *objects* are uniformly and inductively—that is, "effectively"— constructed as a *type system* $\mathcal{M}^L$. In fact, $\mathcal{M}^L$ is a partial mapping from the set of closed terms to the set of binary relations in closed terms, and can provide a semantics of ITT. Statements of the following forms:

$$T \text{ type}, \quad T = T', \quad t \in T, \quad \text{and} \quad t = t' \in T$$

will be interpreted, respectively, as

$$T \in \mathrm{dom}(\mathcal{M}^L), \quad \mathcal{M}^L(T) = \mathcal{M}^L(T'), \quad (t,t) \in \mathcal{M}^L(T), \quad \text{and} \quad (t,t') \in \mathcal{M}^L(T).$$

However, to serve as an adequate semantics of ITT, $\mathcal{M}^L$ has to have the property called *extensionality*. The construction is done in a more uniform way than in [3, 4] by making the best use of the kinds of the type constructors introduced in the pretype-system part of the method system. The validity of the inductive definition of $\mathcal{M}^L$ is guaranteed by this "kind-based" construction, which is intrinsically due to *predicativity* in ITT. (Details are presented in Section 4.)

The main theorem in this paper uses these notions and shows that the semantical anticipation and validity persistency conditions hold for ITT; that is, the original inference rules given in [14] remain valid in any type system built from a *deterministic* and *regular* extension of the original method system that includes all the type constructors explicitly presented in [14]. Determinism guarantees the uniqueness of the result of evaluating a term, and regularity guarantees that the binary relation generated by the defining expression of each type constructor will be a partial equivalence relation in closed terms.

From a practical viewpoint, this result includes a prescription for the class of types that can be introduced into the theory; that is, it prescribes that the class should consist of types *representable* in some type system built from a deterministic and regular extension of the original method system. We can use this criterion to determine whether new types under consideration can be introduced.

1.4 Comparison with Related Work

This paper's result is a natural generalization of Allen's non-type-theoretical reinterpretation of ITT [3, 4], and it demonstrates a structural extension of Howe's formulation of computational open-endedness [12, 13].[3] In fact, Allen's reinterpretation, which is closely related to such realizability-like interpretations as in [7, 20, 11], is exactly the same as building the type system from the original method system. And Howe's formulation would be obtained by restricting the present formulation to extensions of the computational part of the method system. Since

[3] Howe presented a language large enough to create a classical model of ITT in which schemas for the law of the excluded middle are valid. This result corresponds to the fact that the BHK-interpretation serves as a classical semantics when methods are understood to mean set-theoretical functions.

the notion of pretype system is created as the extensible structural part of a method system, open-endedness of types is also well-formulated here for the first time. It is notable that this paper's result is more explicit and comprehensive than that in [4] because it clarifies the relevant underlying conditions and includes a prescription for the class of types that can be introduced into the theory. This class not only includes all the canonical types formed by the type constructors originally presented by Martin-Löf [14], but also seems to be large enough for practical applications.

In [17, 2], a systematic approach was developed to analyze a series of extended languages and their semantics: the target example was the Logical Theory of Constructions (LTC). An "effective" procedure for extending the semantics as the language advances, however, has not been presented.[4] The formulation presented here for ITT provides such a procedure.

A general way of introducing types is also developed in [10], and this way is related to the treatment of types in [5, 9]. Although the method in [10] is developed for a more rigid variant of ITT based on "typed" languages,[5] it gives useful rules for new types, whereas the present paper gives a semantical condition. This semantical characterization of when a type can be sensibly added, however, is obtained as a corollary of the main result, which is the complete demonstration of semantical anticipation and validity persistency in ITT under the concrete construction of type systems. This is very different from the presentation of rather direct set-theoretical semantics in [10].

2 Method Systems

A *method system* is denoted by a pair $L = (C, S)$, where C is a preobject system and S is a pretype system on C.

2.1 Computational Part: Preobject Systems

A *preobject system*[6] is denoted by a quadruple $C = (O, K, \alpha, R)$, where O is a set of *operators*, $K \subset O$ is a set of *canonical* operators, α is a function from O to $\{(k_1, \ldots, k_n) \mid n, k_i \geq 0\}$, and R is a set of *evaluation rules* that will be described later in this subsection. For $\rho \in O$, $\alpha(\rho)$ is the *arity* of ρ and specifies the number and binding structure of the operator's arguments.

For each $m \geq 0$, fix an infinite set of variables called the *second-order variables of arity* m. In particular, a second-order variable is called a *variable* if its arity is 0; otherwise it is called a *metavariable*. The *second-order term schemas of arity* m are inductively defined as follows:

1. a second-order variable of arity m is a second-order term schema of arity m;
2. if ρ is an operator with arity $(k_1, \ldots, k_n)$, if $c_1, \ldots, c_n$ are second-order term schemas of arity $k_1, \ldots, k_n$ respectively, and if $\overline{x}$ is a list of m distinct variables, then $\overline{x}.\rho(c_1, \ldots, c_n)$ is a second-order term schema of arity m;
3. if b is a second-order term schema of arity n, if all of $a_1, \ldots, a_n$ are second-order term schemas of arity 0, and if $\overline{x}$ is a list of m distinct variables, then $\overline{x}.b(a_1, \ldots, a_n)$ is a second-order term schema of arity m.

[4] This seems to be due to LTC's lack of semantical explanation.

[5] For example, [10] does not treat such "valid" statements as $\mathrm{E}(\mathrm{Ap}(t, t), xy.0) = 0 \in \mathrm{N}$, where $t \equiv \lambda(z.(\mathrm{Ap}(z, z), 0))$. In fact, the present paper deals with richer languages by dividing a language into the "untyped" computational part and the more rigid structural part.

[6] Since a preobject system is the same as a *structured lazy computation system* developed by Howe [12, 13], this paper basically follows his definition.

In particular, the *term schemas* are the second-order term schemas of arity 0. The *second-order terms of arity* m are the second-order term schemas of arity m including no metavariables, and in particular, the *terms* are the second-order terms of arity 0. A *simple term schema* is a term schema of the form $\rho(\overline{c})$, where ρ is an operator and $\overline{c}$ is a list of distinct second-order variables. A term or a simple term schema of the form $\rho(\overline{c})$ is *canonical* if ρ is a canonical operator; otherwise it is *noncanonical.*

Binding structure can be added by specifying that in a second-order term schema $x_1, \ldots, x_m.b$ of arity m, each x_i binds in b. And second-order term schemas are assumed to be identical up to renaming of bound variables. Moreover, if $x_1, \ldots, x_m.b$ is a second-order term schema of arity m and if $a_1, \ldots, a_m$ is a list of term schemas, then $(x_1, \ldots, x_m.b)(a_1, \ldots, a_m)$ is identified with the result of simultaneous substituting in the term schema b the term schemas $a_1, \ldots, a_m$ for free variables $x_1, \ldots, x_m$ respectively. The set of closed second-order terms of arity m is denoted by $\widehat{T}^m(C)$. In particular, the set of closed terms is denoted by $\widehat{T}(C)$.[7]

A *valuation* is a map that assigns an arbitrary element of $\widehat{T}^m(C)$ to each second-order variable of arity m. The domain of a valuation is naturally extended to the set of second-order term schemas.

Consider now a set $R = \{r_i\}_{i \in I}$ of evaluation rules, where each r_i has the form

$$a \to b \Leftarrow a_1 \to b_1 \;\&\; \cdots \;\&\; a_n \to b_n \quad (n \geq 1),^{8}$$

and satisfies the following conditions: (1) a is a noncanonical simple term schema; (2) each b_i $(1 \leq i \leq n)$ is a variable or a canonical simple term schema and has neither variables nor metavariables in common with a or b_j $(j \neq i)$; (3) each a_i $(1 \leq i \leq n)$ is a term schema whose free variables and metavariables must occur in a or b_j $(j < i)$; and (4) b is the same variable as b_n.

Let R^+ be the set $\{a \to a \mid a \text{ is a canonical simple term schema}\} \cup R$. The *evaluation relation* $\to_C \subset \widehat{T}(C) \times \widehat{T}(C)$ is defined by the inductive definition: if V is a valuation, if $a \to b \Leftarrow a_1 \to b_1 \;\&\; \cdots \;\&\; a_n \to b_n$ is in R^+, and if $V(a_i) \to_C V(b_i)$ for every i, then $V(a) \to_C V(b)$.

Lemma 2.1 The evaluation relation $\to_C$ of a preobject system C has the following property:

1. if $t \to_C s$, then s is canonical, and
2. if s is a closed canonical term, then $s \to_C u$ if and only if $s \equiv u$.

Proof. Immediate from the definition of $\to_C$. ∎

This property shows that the evaluation is *lazy.* The evaluation relation is *deterministic* if for every t, s, and u, $t \to_C s$ and $t \to_C u$ imply $s \equiv u$. A preobject system is *deterministic* if the evaluation relation produced by its evaluation rules is deterministic, and a method system is *deterministic* if its preobject system is deterministic.

2.2 Structural Part: Pretype Systems

The following specifications will be necessary to introduce a canonical type:

- a type constructor that, with its arguments, will form the canonical type;
- a prescription for the equality relation in the canonical type.

[7] In general, $a, b, c, \ldots$ are syntactical variables that vary through second-order term schemas; $s, t, u, \ldots$ are syntactical variables that vary through closed second-order terms; and $x, y, z, \ldots$ are syntactical variables that vary through variables.

[8] More generally, the set of premises of a rule can be extended to an infinite set providing it is well-ordered.

These are specified as a *pretype system* $S = (D, \kappa, \varphi)$ defined on a preobject system $C = (O, K, \alpha, R)$.

2.2.1 Type Constructors and Their Kinds

A set of *type constructors* is specified as $D \subset K - \{\mathrm{U}_n \mid n \geq 0\}$, where the fixed set $\{\mathrm{U}_n \mid n \geq 0\}$ of *universes* is supposed to be included in the set of canonical operators of every preobject system. A function[9]

$$\kappa : D \to \left(\bigcup_{m \geq 1} \{1\} \{\square\}^* \{2\} \{\square\}^* \{3\} \{\square\}^* \cdots \{m\} \{\square\}^* \right)^*$$

gives the *kind* $\kappa(\Delta)$ of Δ for each $\Delta \in D$.

Some basic operations associated with kinds are introduced: if

$$\kappa(\Delta) = \overbrace{1 \underbrace{\square \cdots \square}_{k_{1,1}\square\text{'s}} \cdots m_1 \underbrace{\square \cdots \square}_{k_{1,m_1}\square\text{'s}}}^{} \cdots \overbrace{1 \underbrace{\square \cdots \square}_{k_{n,1}\square\text{'s}} \cdots m_n \underbrace{\square \cdots \square}_{k_{n,m_n}\square\text{'s}}}^{}$$

for $\Delta \in D$, then

$$|\kappa(\Delta)| = n, \quad \kappa(\Delta)_i = m_i, \quad \kappa(\Delta)_{i,j} = k_{i,j}$$

for every i and j such that $1 \leq i \leq n$ and $1 \leq j \leq m_i$, and

$$\beta(\kappa(\Delta)) = (\overbrace{\underbrace{0, 0, \ldots, 0}_{k_{1,1}0\text{'s}}, \ldots, \underbrace{m_1{-}1, m_1{-}1, \ldots, m_1{-}1}_{k_{1,m_1}m_1-1\text{'s}}}^{}, \ldots, \overbrace{\underbrace{0, 0, \ldots, 0}_{k_{n,1}0\text{'s}}, \ldots, \underbrace{m_n{-}1, \ldots, m_n{-}1}_{k_{n,m_n}m_n-1\text{'s}}}^{}).$$

In ITT, any of the type constructors can be regarded as a map that assigns a new canonical type to bundles of "families of types" and associated "functions," and the notion of kind is abstracted from the domains of such maps. For example, the kinds of the basic type constructors are listed in Table 1.

Table 1: Kinds of the basic type constructors.

Kind	ε	$1\square\square$	11	12
Type Constructor	N_n, N	I	+	Π, Σ, W

More generally, a type constructor Δ will assign a new canonical type to $|\kappa(\Delta)|$ bundles of "families of types" and associated "functions":

- for each i such that $1 \leq i \leq |\kappa(\Delta)|$, the i-th bundle has $\kappa(\Delta)_i$ layers of families of types;
- for each i and j such that $1 \leq i \leq |\kappa(\Delta)|$ and $1 \leq j \leq \kappa(\Delta)_i$, the j-th layer family of types in the i-th bundle is accompanied by $\kappa(\Delta)_{i,j}$ associated functions.

For the sake of the compatibility of kinds with arities, impose the following condition on κ: for every $\Delta \in D$,

$$\beta(\kappa(\Delta)) = \alpha(\Delta).$$

[9]When S and S' are sets of symbols, the concatenation of S and S' is denoted by SS', and the Kleene closure of S is denoted by S^*.

Because of this condition, a canonical type formed by $\Delta \in D$ will have the form

$$\Delta((T_{i,j}, t_{i,j,1}, \ldots, t_{i,j,\kappa(\Delta)_{i,j}})_{\substack{1 \leq i \leq |\kappa(\Delta)| \\ 1 \leq j \leq \kappa(\Delta)_i}}),$$

where all of $T_{i,j}, t_{i,j,1}, \ldots, t_{i,j,\kappa(\Delta)_{i,j}}$ are closed second-order terms of arity $j-1$ for every i and j such that $1 \leq i \leq |\kappa(\Delta)|$ and $1 \leq j \leq \kappa(\Delta)_i$.

2.2.2 Prescriptions for Equality Relations

The main difficulty in defining pretype systems is to present a uniform mechanism by which we can prescribe a large variety of equality relations for various canonical types. Such uniformity will be necessary to create "effective" semantics for a variety of extended languages. To achieve uniformity and effectiveness, this paper develops a systematical way to generate a certain large class $\mathcal{E}(\Delta)$ of expressions from a type constructor Δ in such a way that any of them may be a prescription for the equality relation in a canonical type formed by Δ. Now a function

$$\varphi : \left(\prod \Delta \in D\right) \mathcal{E}(\Delta)$$

is introduced. To specify $\mathcal{E}(\Delta)$ for each $\Delta \in D$, we need a preliminary formal language. When V is an arbitrary set of second-order variables, the $\mathcal{E}(\Delta)_V$-*terms* are inductively defined as follows:

1. a variable is an $\mathcal{E}(\Delta)_V$-term;
2. if $c \in V$ is a second-order variable of arity m and if all of $a_1, \ldots, a_m$ are $\mathcal{E}(\Delta)_V$-terms, then $c(a_1, \ldots, a_m)$ is an $\mathcal{E}(\Delta)_V$-term;
3. if i and j are such that $1 \leq i \leq |\kappa(\Delta)|$ and $1 \leq j \leq \kappa(\Delta)_i$ and if all of $a_1, \ldots, a_{j-1}$ are $\mathcal{E}(\Delta)_V$-terms, then $F_{i,j,1}(a_1, \ldots, a_{j-1}), \ldots, F_{i,j,\kappa(\Delta)_{i,j}}(a_1, \ldots, a_{j-1})$ are also $\mathcal{E}(\Delta)_V$-terms, where all of $F_{i,j,1}, \ldots, F_{i,j,\kappa(\Delta)_{i,j}}$ are new symbols.

In particular, the $\widehat{\mathcal{E}}(\Delta)_V$-*terms* are the $\mathcal{E}(\Delta)_V$-terms including no variables other than those in V.

The $\mathcal{E}(\Delta)_V$-*formulas* are inductively defined as follows:

1. if a_1 and a_2 are $\mathcal{E}(\Delta)_V$-terms, then $P(a_1, a_2)$ is an $\mathcal{E}(\Delta)_V$-formula, where P is a new symbol;
2. if i and j are such that $1 \leq i \leq |\kappa(\Delta)|$ and $1 \leq j \leq \kappa(\Delta)_i$ and if all of $a_1, \ldots, a_{j+1}$ are $\mathcal{E}(\Delta)_V$-terms, then $Q_{i,j}(a_1, \ldots, a_{j+1})$ is an $\mathcal{E}(\Delta)_V$-formula, where $Q_{i,j}$ is a new symbol;
3. if A and B are $\mathcal{E}(\Delta)_V$-formulas, then $\bot$, $A \,\&\, B$, $A \vee B$, and $A \Rightarrow B$ are also $\mathcal{E}(\Delta)_V$-formulas;
4. if v is a variable other than those in V and if A is an $\mathcal{E}(\Delta)_V$-formula, then $\forall v.A$ and $\exists v.A$ are also $\mathcal{E}(\Delta)_V$-formulas.

In addition, impose the following conditions on an $\mathcal{E}(\Delta)_V$-formula:

1. its free variables must be among the variables in V;
2. *strictly positive condition*: every occurrence of P must be *strictly positive*;[10]
3. *contextual condition*: for every i and j such that $1 \leq i \leq |\kappa(\Delta)|$ and $1 \leq j \leq \kappa(\Delta)_i$, if $Q_{i,j}(a_1, \ldots, a_{j-1}, a_j, a_{j+1})$ occurs in the formula, then for every k such that $1 \leq k \leq j-1$, there exists its subformula $A_k \Rightarrow B_k$ such that
   (a) B_k includes the occurrence of $Q_{i,j}(a_1, \ldots, a_{j-1}, a_j, a_{j+1})$ and no variable in a_k is bound in B_k;

(b) A_k has the form $Q_{i,k}(b_1, \ldots, b_{k-1}, a_k, a'_k)$ or $Q_{i,k}(b_1, \ldots, b_{k-1}, a'_k, a_k)$.

Let $\theta_1 \cdots \theta_h$ be a string of distinct canonical operators in $K - (D \cup \{\mathrm{U}_n | n \geq 0\})$, and suppose that e and e' are $\widehat{\mathcal{E}}(\Delta)_V$-terms. An $\mathcal{E}(\Delta)_V^{\theta_1 \cdots \theta_h}(e, e')$-*expression* has the form

$$\begin{array}{c} \exists \overline{c_1} \exists \overline{c_1}'.e \to \theta_1(\overline{c_1}) \;\&\; e' \to \theta_1(\overline{c_1}') \;\&\; A_1 \\ \vee \cdots \vee \\ \exists \overline{c_h} \exists \overline{c_h}'.e \to \theta_h(\overline{c_h}) \;\&\; e' \to \theta_h(\overline{c_h}') \;\&\; A_h, \end{array}$$

where for each l such that $1 \leq l \leq h$,

1. $\theta_l(\overline{c_l})$ and $\theta_l(\overline{c_l}')$ are canonical simple term schemas such that all of the second-order variables in $\overline{c_l}$ or $\overline{c_l}'$ are distinct and other than those in V, and
2. A_l is either
   (a) an $\mathcal{E}(\Delta)_{V_l}$-formula, or
   (b) an $\mathcal{E}(\Delta)_{V_l}^{\eta_1 \cdots \eta_k}(f, f')$-expression such that $\eta_1 \cdots \eta_k$ is a string of distinct canonical operators in $K - (D \cup \{\mathrm{U}_n | n \geq 0\})$, and both f and f' are $\widehat{\mathcal{E}}(\Delta)_{V_l}$-terms,

   where V_l is the result of removing from $V \cup \{\overline{c_l}, \overline{c_l}'\}$ all the second-order variables in e or e'.

An $\mathcal{E}(\Delta)$-*expression* has the form

$$\mu P.(x, x').\Psi,$$

where x and x' are distinct variables and Ψ is an $\mathcal{E}(\Delta)_{\{x,x'\}}^{\theta_1 \cdots \theta_h}(x, x')$-expression for some string $\theta_1 \cdots \theta_h$ of distinct canonical operators in $K - (D \cup \{\mathrm{U}_n | n \geq 0\})$. In this case, $\theta_1 \cdots \theta_h$ is called the *canonical index* of the $\mathcal{E}(\Delta)$-expression.

Let $\mathcal{E}(\Delta)$ be the set of $\mathcal{E}(\Delta)$-expressions. To guarantee a certain non-overlapping property, impose the following conditions on φ: for every $\Delta, \Delta' \in D$, the canonical indices of $\varphi(\Delta)$ and $\varphi(\Delta')$ have no canonical operators in common.

By using valuations, an $\mathcal{E}(\Delta)$-expression ψ of the form $\mu P.(x, x').\Psi[P, x, x']$ will be interpreted as the least relation $[\![\psi]\!] \subset \widehat{T}(C) \times \widehat{T}(C)$ such that

$$[\![\psi]\!](t, t') \Leftrightarrow [\![\Psi]\!]([\![\psi]\!], t, t'),$$

where $[\![\Psi]\!]$ is the standard interpretation of Ψ under an appropriate translation of the parametric symbols $Q_{i,j}, F_{i,j,1}, \ldots, F_{i,j,\kappa(\Delta)_{i,j}}$ in Ψ. Note that standard intuitionistic theory of inductive definitions directly allows the definition of $[\![\psi]\!]$; the existence of such $[\![\psi]\!]$ is guaranteed by the strictly positive condition on Ψ. See [9, 10, 21, 22, 19] for the development of the theory of inductive definitions.

3 Extensions of Method Systems

A language that includes all the type constructors explicitly presented in [14] is specified as the original method system $L_0 = (C_0, S_0)$.

Lemma 3.1 The method system L_0 is deterministic.

Proof. Induction on elements in $\to_{C_0}$ shows that for every t and s such that $t \to_{C_0} s$, $t \to_{C_0} u$ implies $s \equiv u$ for every u. ■

Table 2: Method System $L_\diamond = ((O_\diamond, K_\diamond, \alpha_\diamond, R_\diamond), (D_\diamond, \kappa_\diamond, \varphi_\diamond))$.

$$O_\diamond = \{\mathrm{N}, 0, \mathrm{S}, \mathrm{R}, \Pi, \lambda, \mathrm{Ap}, \mathrm{U}_n \text{ (for every } n \geq 0)\},$$
$$K_\diamond = \{\mathrm{N}, 0, \mathrm{S}, \Pi, \lambda, \mathrm{U}_n \text{ (for every } n \geq 0)\},$$
$$\alpha_\diamond(\rho) = \begin{cases} () & \text{if } \rho \text{ is N, 0, U}_n \text{ for } n \geq 0, \\ (0) & \text{if } \rho \text{ is S}, \\ (1) & \text{if } \rho \text{ is } \lambda, \\ (0,0) & \text{if } \rho \text{ is Ap}, \\ (0,1) & \text{if } \rho \text{ is } \Pi, \\ (0,0,2) & \text{if } \rho \text{ is R}, \end{cases}$$
$$R_\diamond = \left\{ \begin{array}{lll} \mathrm{R}(c,d,e) \to f & \Leftarrow & c \to 0 \,\&\, d \to f, \\ \mathrm{R}(c,d,e) \to f & \Leftarrow & c \to \mathrm{S}(a) \,\&\, e(a, \mathrm{R}(a,d,e)) \to f, \\ \mathrm{Ap}(c,a) \to d & \Leftarrow & c \to \lambda(b) \,\&\, b(a) \to d \end{array} \right\},$$
$$D_\diamond = \{\mathrm{N}, \Pi\},$$
$$\kappa_\diamond(\Delta) = \begin{cases} \varepsilon & \text{if } \Delta \text{ is N}, \\ 12 & \text{if } \Delta \text{ is } \Pi, \end{cases}$$
$$\varphi_\diamond(\Delta) = \begin{cases} \mu P.(x,x').\ x \to 0 \,\&\, x' \to 0 \ \vee & \text{if } \Delta \text{ is N}, \\ \qquad \exists a \exists a'.\ x \to \mathrm{S}(a) \,\&\, x' \to \mathrm{S}(a') \,\&\, P(a,a') & \\ \mu P.(x,x').\ \exists b \exists b'.\ x \to \lambda(b) \,\&\, x' \to \lambda(b') & \text{if } \Delta \text{ is } \Pi. \\ \qquad \&\, \forall v \forall v'.\ Q_{1,1}(v,v') \Rightarrow Q_{1,2}(v, b(v), b'(v')) & \end{cases}$$

The complete description of L_0 is given in Appendix A. Table 2 describes only the subsystem $L_\diamond = ((O_\diamond, K_\diamond, \alpha_\diamond, R_\diamond), (D_\diamond, \kappa_\diamond, \varphi_\diamond))$, which includes *natural numbers*, *Cartesian product of a family of types*, and *universes*.

Let $L = ((O, K, \alpha, R), (D, \kappa, \varphi))$ and $L' = ((O', K', \alpha', R'), (D', \kappa', \varphi'))$ be arbitrary method systems. If the following conditions hold, L' is called an *extension* of L:

- $O \subset O'$;
- $K = K' \cap O$;
- $\alpha = \alpha' \mid O$;
- $\forall r\, [r \in R \Leftrightarrow r \in R' \ \&\ o(r) \in O]$,
  where if r has the form $\rho(\overline{c}) \to b \Leftarrow a_1 \to b_1 \,\&\, \cdots \,\&\, a_n \to b_n$, then $o(r) = \rho$;
- $D \subset D'$;
- $\kappa = \kappa' \mid D$;
- $\varphi = \varphi' \mid D$;

where $f \mid Z$ denotes the function $f : X \to Y$ whose domain is restricted to $Z \subset X$. It is easy to see that the binary relation induced by the notion of extension is a partial ordering of method systems. From the computational viewpoint, the following lemma holds:

Lemma 3.2 [computational conservativeness] Suppose that $L' = (C', S')$ is an extension of $L = (C, S)$. For every t and s such that $t \to_{C'} s$, $t \in \widehat{T}(C)$ implies $t \to_C s$.

Proof. Suppose that $C' = (O', K', \alpha', R')$ and $C = (O, K, \alpha, R)$. The lemma is proved by induction on elements in $\to_{C'}$. It is trivial when t is canonical, so assume that t and

[10] See [23], for example.

Table 3: Method System $L_\sqcap = ((O_\sqcap, K_\sqcap, \alpha_\sqcap, R_\sqcap), (D_\sqcap, \kappa_\sqcap, \varphi_\sqcap))$.

$$O_\sqcap = O_\diamond \cup \{\sqcap, \iota, \mathrm{B}\}, \quad K_\sqcap = K_\diamond \cup \{\sqcap, \iota\},$$

$$\alpha_\sqcap(\rho) = \begin{cases} (0,0) & \text{if } \rho \text{ is } \sqcap, \\ (0) & \text{if } \rho \text{ is } \iota, \\ (0,1) & \text{if } \rho \text{ is B}, \\ \alpha_\diamond(\rho) & \text{otherwise}, \end{cases}$$

$$R_\sqcap = R_\diamond \cup \{\mathrm{B}(c,d) \to e \Leftarrow c \to \iota(a) \ \&\ d(a) \to e\},$$

$$D_\sqcap = D_\diamond \cup \{\sqcap\},$$

$$\kappa_\sqcap(\Delta) = \begin{cases} 11 & \text{if } \Delta \text{ is } \sqcap, \\ \kappa_\diamond(\Delta) & \text{otherwise}, \end{cases}$$

$$\varphi_\sqcap(\Delta) = \begin{cases} \mu P.(x,x').\ \exists a \exists a'.\ x \to \iota(a) \ \&\ x' \to \iota(a') \\ \qquad\qquad \&\ Q_{1,1}(a,a') \ \&\ Q_{2,1}(a,a') & \text{if } \Delta \text{ is } \sqcap, \\ \varphi_\diamond(\Delta) & \text{otherwise.} \end{cases}$$

s are respectively written as $V(\rho(\overline{c}))$ and $V(b)$, for an evaluation rule $r \in R'$ of the form $\rho(\overline{c}) \to b \Leftarrow a_1 \to b_1 \ \& \cdots \& a_n \to b_n$ and for a valuation V. Since $t \in \widehat{T}(C)$, ρ must be in O; hence $r \in R$. So it suffices to prove that $V(a_i) \to_C V(b_i)$ for every i. This follows immediately from induction hypothesis and the definition of evaluation rules. ■

Along the lines mentioned above, various extensions of $L_\diamond$ can be built to obtain richer languages. The method system L_0 is a deterministic example of such an extension, and a large class of inductively defined types can be incorporated successively. Some other examples follow.

Example 3.3 The method system $L_\sqcap$, which is the deterministic extension of $L_\diamond$ with "intersection" types, is described in Table 3.

Example 3.4 The method system $L_\sqcup$, which is the deterministic extension of $L_\diamond$ with "union" types, is described in Table 4.

Example 3.5 The method system $L_\subset$, which is the deterministic extension of $L_\diamond$ with "subset" types, is described in Table 5.

Example 3.6 The method system $L_\star$, which is the nondeterministic extension of $L_\diamond$ with the "Amb" operator, is described in Table 6.

4 Type Systems

This section presents a semantics of ITT whose underlying language is specified as a method system. Let $L = (C, S)$ be an arbitrary method system, where $C = (O, K, \alpha, R)$ is a preobject system and $S = (D, \kappa, \varphi)$ is a pretype system on C.

[11] In [1], the abbreviation $A \ \&\Rightarrow B$ is used for $A \ \& [A \Rightarrow B]$. Also see the descriptions of $\varphi_0(\Sigma)$ and $\varphi_0(\mathrm{W})$ in Appendix A.

Table 4: Method System $L_{\sqcup} = ((O_{\sqcup}, K_{\sqcup}, \alpha_{\sqcup}, R_{\sqcup}), (D_{\sqcup}, \kappa_{\sqcup}, \varphi_{\sqcup}))$.

$$O_{\sqcup} = O_{\diamond} \cup \{\sqcup, \nu, \mathrm{A}\}, \quad K_{\sqcup} = K_{\diamond} \cup \{\sqcup, \nu\},$$

$$\alpha_{\sqcup}(\rho) = \begin{cases} (0,0) & \text{if } \rho \text{ is } \sqcup, \\ (0) & \text{if } \rho \text{ is } \nu, \\ (0,1) & \text{if } \rho \text{ is A}, \\ \alpha_{\diamond}(\rho) & \text{otherwise}, \end{cases}$$

$$R_{\sqcup} = R_{\diamond} \cup \{\mathrm{A}(c,d) \to e \Leftarrow c \to \nu(a) \ \&\ d(a) \to e\},$$

$$D_{\sqcup} = D_{\diamond} \cup \{\sqcup\},$$

$$\kappa_{\sqcup}(\Delta) = \begin{cases} 11 & \text{if } \Delta \text{ is } \sqcup, \\ \kappa_{\diamond}(\Delta) & \text{otherwise}, \end{cases}$$

$$\varphi_{\sqcup}(\Delta) = \begin{cases} \mu P.(x,x').\ \exists a \exists a'.\ x \to \nu(a) \ \&\ x' \to \nu(a') & \text{if } \Delta \text{ is } \sqcup, \\ \qquad\qquad \&\ [Q_{1,1}(a,a') \vee Q_{2,1}(a,a')] & \\ \varphi_{\diamond}(\Delta) & \text{otherwise.} \end{cases}$$

Table 5: Method System $L_{\mathsf{C}} = ((O_{\mathsf{C}}, K_{\mathsf{C}}, \alpha_{\mathsf{C}}, R_{\mathsf{C}}), (D_{\mathsf{C}}, \kappa_{\mathsf{C}}, \varphi_{\mathsf{C}}))$.

$$O_{\mathsf{C}} = O_{\diamond} \cup \{\{\cdot|\cdot\}, \mathrm{sub}, \mathrm{L}\}, \quad K_{\mathsf{C}} = K_{\diamond} \cup \{\{\cdot|\cdot\}, \mathrm{sub}\},$$

$$\alpha_{\mathsf{C}}(\rho) = \begin{cases} (0,1) & \text{if } \rho \text{ is } \{\cdot|\cdot\}, \\ (0) & \text{if } \rho \text{ is sub}, \\ (0,1) & \text{if } \rho \text{ is L}, \\ \alpha_{\diamond}(\rho) & \text{otherwise}, \end{cases}$$

$$R_{\mathsf{C}} = R_{\diamond} \cup \{\mathrm{L}(c,d) \to e \Leftarrow c \to \mathrm{sub}(a) \ \&\ d(a) \to e\},$$

$$D_{\mathsf{C}} = D_{\diamond} \cup \{\{\cdot|\cdot\}\},$$

$$\kappa_{\mathsf{C}}(\Delta) = \begin{cases} 12 & \text{if } \Delta \text{ is } \{\cdot|\cdot\}, \\ \kappa_{\diamond}(\Delta) & \text{otherwise}, \end{cases}$$

$$\varphi_{\mathsf{C}}(\Delta) = \begin{cases} \mu P.(x,x').\ \exists a \exists a'.\ x \to \mathrm{sub}(a) \ \&\ x' \to \mathrm{sub}(a') & \text{if } \Delta \text{ is } \{\cdot|\cdot\}, \\ \qquad\qquad \&\ Q_{1,1}(a,a') & \\ \qquad\qquad \&\ Q_{1,1}(a,a') \Rightarrow [\exists b.Q_{1,2}(a,b,b)\ \&\ \exists b'.Q_{1,2}(a',b',b')]^{11} & \\ \varphi_{\diamond}(\Delta) & \text{otherwise.} \end{cases}$$

Table 6: Method System $L_{\star} = ((O_{\star}, K_{\star}, \alpha_{\star}, R_{\star}), (D_{\star}, \kappa_{\star}, \varphi_{\star}))$.

$$O_{\star} = O_{\diamond} \cup \{\mathrm{Amb}\}, \quad K_{\star} = K_{\diamond},$$

$$\alpha_{\star}(\rho) = \begin{cases} (0,0) & \text{if } \rho \text{ is Amb}, \\ \alpha_{\diamond}(\rho) & \text{otherwise}, \end{cases}$$

$$R_{\star} = R_{\diamond} \cup \{\mathrm{Amb}(a,b) \to c \Leftarrow a \to c,\ \mathrm{Amb}(a,b) \to c \Leftarrow b \to c\},$$

$$D_{\star} = D_{\diamond}, \quad \kappa_{\star} = \kappa_{\diamond}, \quad \varphi_{\star} = \varphi_{\diamond}.$$

4.1 Inductive Definitions of Type Systems

A *type system* τ over L is a subset of $\widehat{T}(C) \times 2^{\widehat{T}(C)\times\widehat{T}(C)}$. In this subsection, a certain type system over L is inductively constructed as the least fixed point of $\mathcal{I}^L(\sigma,\cdot)$, where $\mathcal{I}^L$ is a binary function on type systems over L, and σ is an appropriate base. Intuitively, $\mathcal{I}^L(\sigma,\tau)$ is the type system that consists of (1) types in σ, and (2) types constructed by type constructors in L with bundles of "families of types" and associated "functions" in τ. More formally, it is defined by

$$\mathcal{I}^L(\sigma,\tau)(T,\phi) \Leftrightarrow \sigma(T,\phi) \vee \mathcal{K}^L(\tau)(T,\phi);$$

$$\begin{aligned}\mathcal{K}^L(\tau)(T,\phi) \Leftrightarrow \exists\Delta \in D\ \exists\Omega\ \exists\Phi.\ & T \to_C \Delta(\Omega)\\ & \& \mathrm{Bun}^L_{\kappa(\Delta)}(\tau,\Omega,\Phi)\\ & \& \forall t\forall t'\left[\phi(t,t') \Leftrightarrow [\![\varphi(\Delta)]\!]^L_{\Omega,\Phi}(t,t')\right].\end{aligned}$$

Here Ω has the form of a sequence

$$(T_{i,j}, t_{i,j,1}, \ldots, t_{i,j,\kappa(\Delta)_{i,j}})_{\substack{1\le i\le|\kappa(\Delta)|\\ 1\le j\le\kappa(\Delta)_i}}$$

such that all of $T_{i,j}, t_{i,j,1}, \ldots, t_{i,j,\kappa(\Delta)_{i,j}}$ are closed second-order terms of arity $j-1$ for every i and j, and Φ has the form of a sequence

$$(\phi_{i,j})_{\substack{1\le i\le|\kappa(\Delta)|\\ 1\le j\le\kappa(\Delta)_i}}$$

such that $\phi_{i,j}$ is a $(j+1)$-ary relation in $\widehat{T}(C)$ for every i and j. Intuitively, $\mathrm{Bun}^L_{\kappa(\Delta)}(\tau,\Omega,\Phi)$ represents Ω with Φ to be bundles of "extensional families of types" and associated "extensional functions" in τ. The precise definition of $\mathrm{Bun}^L_{\kappa(\Delta)}$ is given by

$$\mathrm{Bun}^L_{\kappa(\Delta)}(\tau,\Omega,\Phi) \Leftrightarrow \mathop{\&}_{\substack{1\le i\le|\kappa(\Delta)|\\ 1\le j\le\kappa(\Delta)_i}} \left[\begin{array}{l}\forall u_1\cdots\forall u_{j-1}\forall u'_1\cdots\forall u'_{j-1}.\\ \phi_{i,1}(u_1,u'_1)\ \&\ \cdots\ \&\ \phi_{i,j-1}(\overline{u},u_{j-1},u'_{j-1})\\ \quad\Rightarrow \left[\begin{array}{l}\tau(T_{i,j}(\overline{u},u_{j-1}),\phi_{i,j}(\overline{u},u_{j-1}))\\ \&\ \tau(T_{i,j}(\overline{u}',u'_{j-1}),\phi_{i,j}(\overline{u}',u'_{j-1}))\\ \&\ \forall s\forall s'[\phi_{i,j}(\overline{u},u_{j-1},s,s') \Leftrightarrow \phi_{i,j}(\overline{u}',u'_{j-1},s,s')]\\ \&\ \phi_{i,j}(\overline{u},u_{j-1},t_{i,j,1}(\overline{u},u_{j-1}),t_{i,j,1}(\overline{u}',u'_{j-1}))\\ \quad\cdots\\ \&\ \phi_{i,j}(\overline{u},u_{j-1},t_{i,j,\kappa(\Delta)_{i,j}}(\overline{u},u_{j-1}),t_{i,j,\kappa(\Delta)_{i,j}}(\overline{u}',u'_{j-1}))\end{array}\right]\end{array}\right],$$

where $\overline{u}$ and $\overline{u}'$ are the abbreviations of the sequences $u_1,\ldots,u_{j-2}$ and $u'_1,\ldots,u'_{j-2}$. Finally, $[\![\varphi(\Delta)]\!]^L_{\Omega,\Phi}$ is the standard interpretation of $\varphi(\Delta)$ under the following translation: (1) for every i and j such that $1 \le i \le |\kappa(\Delta)|$ and $1 \le j \le \kappa(\Delta)_i$, the symbols $F_{i,j,1},\ldots,F_{i,j,\kappa(\Delta)_{i,j}}$ in $\varphi(\Delta)$ are interpreted as the respective second-order terms $t_{i,j,1},\ldots,t_{i,j,\kappa(\Delta)_{i,j}}$ in Ω; (2) for every i and j such that $1 \le i \le |\kappa(\Delta)|$ and $1 \le j \le \kappa(\Delta)_i$, the symbol $Q_{i,j}$ in $\varphi(\Delta)$ is interpreted as the relation $\phi_{i,j}$ in Φ; and (3) the symbol $\to$ in $\varphi(\Delta)$ is interpreted as $\to_C$.

It is easy to see that for each σ, $\mathcal{I}^L(\sigma,\cdot)$ is a monotone function on type systems over L; therefore, the type system $\mu^L(\sigma)$ is well-defined[12] by

$$\mu^L(\sigma)(T,\phi) \Leftrightarrow \forall\tau\left[\mathcal{I}^L(\sigma,\tau) \subset \tau \Rightarrow \tau(T,\phi)\right]$$

as the least fixed point of $\mathcal{I}^L(\sigma,\cdot\,)$. This can be used to define the type systems $\mathcal{S}_n^L$ and $\mathcal{M}_n^L$ for each $n \geq 0$, and also to define the type systems $\mathcal{S}^L$ and $\mathcal{M}^L$:

$$\begin{array}{lcl}
\mathcal{S}_n^L & = & \{(T, =_{\mathcal{M}_m^L}) \mid 0 \leq m < n \;\&\; T \to_C \mathrm{U}_m \\
 & & \;\&\; \forall S \forall S'[S =_{\mathcal{M}_m^L} S' \Leftrightarrow \exists \phi.\mathcal{M}_m^L(S,\phi) \;\&\; \mathcal{M}_m^L(S',\phi)]\}; \\
\mathcal{M}_n^L & = & \mu^L(\mathcal{S}_n^L); \\
\mathcal{S}^L & = & \bigcup_{n\geq 0} \mathcal{S}_n^L; \\
\mathcal{M}^L & = & \mu^L(\mathcal{S}^L).
\end{array}$$

4.2 Semantics of ITT

A *statement* of L has one of the following forms:

$$\begin{array}{ll}
T(x_1,\ldots,x_n) \text{ type} & (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})); \\
T(x_1,\ldots,x_n) = T'(x_1,\ldots,x_n) & (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})); \\
t(x_1,\ldots,x_n) \in T(x_1,\ldots,x_n) & (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})); \\
t(x_1,\ldots,x_n) = t'(x_1,\ldots,x_n) \in T(x_1,\ldots,x_n) & (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1}));
\end{array}$$

where $x_1,\ldots,x_n$ are distinct variables, and $T, T', t, t', T_1,\ldots,T_n$ are closed second-order terms, respectively, of arity $n, n, n, n, 0,\ldots,n-1$. The sequence $x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})$ is called the *hypothesis* of the statement.

The type system $\mathcal{M}^L$ provides a semantics of ITT whose underlying language is specified as L. In fact, the property $\mathcal{M}^L \models \Theta$ of a statement Θ of L, which expresses that Θ is *valid* in $\mathcal{M}^L$, can be defined as follows:

$$\begin{array}{l}
\mathcal{M}^L \models T(x_1,\ldots,x_n) \text{ type } (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})) \\
\Leftrightarrow \exists (\phi_j)_{1\leq j\leq n} \exists \phi.\ \mathrm{Bun}^L_{1\cdots n(n+1)}(\mathcal{M}^L, (T_j)_{1\leq j\leq n}, T, (\phi_j)_{1\leq j\leq n}, \phi);
\end{array}$$

$$\begin{array}{l}
\mathcal{M}^L \models T(x_1,\ldots,x_n) = T'(x_1,\ldots,x_n)\ (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})) \\
\Leftrightarrow \exists (\phi_j)_{1\leq j\leq n} \exists \phi \exists \phi'.\ \mathrm{Bun}^L_{1\cdots n(n+1)}(\mathcal{M}^L, (T_j)_{1\leq j\leq n}, T, (\phi_j)_{1\leq j\leq n}, \phi)\ \& \\
\qquad\qquad \mathrm{Bun}^L_{1\cdots n(n+1)}(\mathcal{M}^L, (T_j)_{1\leq j\leq n}, T', (\phi_j)_{1\leq j\leq n}, \phi')\ \& \\
\qquad\qquad \forall u_1 \cdots \forall u_n.\ \phi_1(u_1,u_1)\ \& \cdots \&\ \phi_n(u_1,\ldots,u_{n-1},u_n,u_n) \\
\qquad\qquad\qquad \Rightarrow \forall s \forall s'[\phi(u_1,\ldots,u_n,s,s') \Leftrightarrow \phi'(u_1,\ldots,u_n,s,s')];
\end{array}$$

$$\begin{array}{l}
\mathcal{M}^L \models t(x_1,\ldots,x_n) \in T(x_1,\ldots,x_n)\ (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})) \\
\Leftrightarrow \exists (\phi_j)_{1\leq j\leq n} \exists \phi.\ \mathrm{Bun}^L_{1\cdots n(n+1)\square}(\mathcal{M}^L, (T_j)_{1\leq j\leq n}, T, t, (\phi_j)_{1\leq j\leq n}, \phi);
\end{array}$$

$$\begin{array}{l}
\mathcal{M}^L \models t(x_1,\ldots,x_n) = t'(x_1,\ldots,x_n) \in T(x_1,\ldots,x_n)\ (x_1 \in T_1,\ldots,x_n \in T_n(x_1,\ldots,x_{n-1})) \\
\Leftrightarrow \exists (\phi_j)_{1\leq j\leq n} \exists \phi.\ \mathrm{Bun}^L_{1\cdots n(n+1)\square}(\mathcal{M}^L, (T_j)_{1\leq j\leq n}, T, t, (\phi_j)_{1\leq j\leq n}, \phi)\ \& \\
\qquad\qquad \mathrm{Bun}^L_{1\cdots n(n+1)\square}(\mathcal{M}^L, (T_j)_{1\leq j\leq n}, T, t', (\phi_j)_{1\leq j\leq n}, \phi)\ \& \\
\qquad\qquad \forall u_1 \cdots \forall u_n.\ \phi_1(u_1,u_1)\ \& \cdots \&\ \phi_n(u_1,\ldots,u_{n-1},u_n,u_n) \\
\qquad\qquad\qquad \Rightarrow \phi(u_1,\ldots,u_n,t(u_1,\ldots,u_n),t'(u_1,\ldots,u_n)).
\end{array}$$

In particular, the definition for a hypothesis-free statement is simply given as follows:

$$\begin{array}{lcl}
\mathcal{M}^L \models T \text{ type} & \Leftrightarrow & \exists \phi.\mathcal{M}^L(T,\phi); \\
\mathcal{M}^L \models T = T' & \Leftrightarrow & \exists \phi.\mathcal{M}^L(T,\phi)\ \&\ \mathcal{M}^L(T',\phi); \\
\mathcal{M}^L \models t \in T & \Leftrightarrow & \exists \phi.\mathcal{M}^L(T,\phi)\ \&\ \phi(t,t); \\
\mathcal{M}^L \models t = t' \in T & \Leftrightarrow & \exists \phi.\mathcal{M}^L(T,\phi)\ \&\ \phi(t,t').
\end{array}$$

[12] As pointed out by Allen [3], the definition of $\mu^L(\sigma)$ is not only set-theoretically valid but probably will be convincing to intuitionists as well because $\mu^L(\sigma)$ may be defined as the least τ such that $\forall T \forall \phi\,[\mathcal{I}^L(\sigma,\tau)(T,\phi) \Rightarrow \tau(T,\phi)]$ and because the relation $\mathcal{I}^L(\sigma,\tau)(T,\phi)$ is strictly positive in τ.

4.3 Extensionality

To serve as an adequate semantics of ITT, a type system τ over L has to satisfy the following conditions:

$$\begin{aligned}
\mathrm{Fun}^L(\tau) &\Leftrightarrow \forall T\forall\phi\forall\phi'\,[\tau(T,\phi)\ \&\ \tau(T,\phi')\Rightarrow\phi=\phi']\,;^{13}\\
\mathrm{TrSy}^L(\tau) &\Leftrightarrow \forall T\forall\phi\,[\tau(T,\phi)\Rightarrow\phi \text{ is transitive and symmetric}]\,;\\
\mathrm{Val}^L(\tau) &\Leftrightarrow \forall T\forall\phi\,[\tau(T,\phi)\Rightarrow\forall t\forall t'\,[\phi(t,t')\Leftrightarrow\exists s.t\to_C s\ \&\ \phi(s,t')]]\,;\\
\mathrm{TyVal}^L(\tau) &\Leftrightarrow \forall T\forall\phi\,[\tau(T,\phi)\Leftrightarrow\exists S.T\to_C S\ \&\ \tau(S,\phi)]\,.
\end{aligned}$$

If τ satisfies these four conditions, it is said to be *extensional.*

If $\mathrm{TrSy}^L(\tau)$ implies $\mathrm{TrSy}^L(\mathcal{K}^L(\tau))$ for every type system τ over L, then L is called *regular.*[14]

Lemma 4.1 The method system L_0 is regular.

Proof. Suppose that $\mathrm{TrSy}^{L_0}(\tau)$. Since L_0 is deterministic, $[\![\varphi_0(\Delta)]\!]^{L_0}_{\Omega,\Phi}$ is transitive as well as symmetric for each $\Delta\in D_0$ and for every Ω and Φ such that $\mathrm{Bun}^{L_0}_{\kappa_0(\Delta)}(\tau,\Omega,\Phi)$. Use induction when Δ is N or W. ■

Among the examples shown in Section 3, $L_\sqcap$ and $L_\subset$ are regular. On the other hand, neither deterministic $L_\sqcup$ nor nondeterministic $L_\star$ satisfies regularity.

The strategy shown in [3] is also applicable to prove the following lemmas. Note that the uniform construction of types in the type systems enables such properties as extensionality to be proved "uniformly" instead of "by case analysis."

Lemma 4.2 [extensionality lemma] If L is a deterministic and regular method system, then $\mathcal{S}^L_n$, $\mathcal{M}^L_n$, $\mathcal{S}^L$, and $\mathcal{M}^L$ are extensional type systems over L for every $n\geq 0$.

Proof. Suppose that $L=(C,S)$, where $C=(O,K,\alpha,R)$ and $S=(D,\kappa,\varphi)$, is deterministic and regular. By virtue of the determinism of L, it is straightforward that (1) for each $n\geq 0$, if $\mathcal{M}^L_m$ is extensional for every m such that $0\leq m<n$, then $\mathcal{S}^L_n$ is also extensional; and (2) if $\mathcal{M}^L_m$ is extensional for every $m\geq 0$, then $\mathcal{S}^L$ is also extensional. Hence it suffices to prove that if σ is an extensional type system over L and if for every T and ϕ, $\sigma(T,\phi)$ implies $T\to_C \mathrm{U}_m$ for some m, then $\mu^L(\sigma)$ is also extensional.

Define two type systems over L by $\tau_{\mathrm{Fun}}=\{(T,\phi)\mid\forall\phi'\,[\mu^L(\sigma)(T,\phi')\Rightarrow\phi=\phi']\}$ and $\tau_{\mathrm{TrSy}}=\{(T,\phi)\mid\phi \text{ is transitive and symmetric}\}$. Then it can be proved that $\mathcal{I}^L(\sigma,\tau_{\mathrm{Fun}})\subset\tau_{\mathrm{Fun}}$ by the determinism of L and by the contextual condition on $\varphi(\Delta)$ for each $\Delta\in D$; therefore, $\mathrm{Fun}^L(\mu^L(\sigma))$ holds. It is also proved that $\mathcal{I}^L(\sigma,\tau_{\mathrm{TrSy}})\subset\tau_{\mathrm{TrSy}}$ by the regularity of L; therefore, $\mathrm{TrSy}^L(\mu^L(\sigma))$ holds. $\mathrm{Val}^L(\mu^L(\sigma))$ and $\mathrm{TyVal}^L(\mu^L(\sigma))$ also hold because $\mathrm{Val}^L(\mathcal{I}^L(\sigma,\tau))$ and $\mathrm{TyVal}^L(\mathcal{I}^L(\sigma,\tau))$ for every τ and because $\mathcal{I}^L(\sigma,\mu^L(\sigma))=\mu^L(\sigma)$. ■

Lemma 4.3 If L is a method system, then $\mathcal{M}^L_n\subset\mathcal{M}^L_{n+1}$ and $\mathcal{M}^L_n\subset\mathcal{M}^L$ for every $n\geq 0$.

Proof. It suffices to prove that $\sigma\subset\sigma'$ implies $\mu^L(\sigma)\subset\mu^L(\sigma')$. Since $\mathcal{I}^L(\sigma',\mu^L(\sigma'))\subset\mu^L(\sigma')$, $\mathcal{I}^L(\sigma,\mu^L(\sigma'))\subset\mu^L(\sigma')$ holds if $\sigma\subset\sigma'$. ■

5 Open-Endedness of Objects and Types

The system of inference rules given in [14] can be formalized as the deduction system ITT_0,[15] which is built up in the language L_0. When L is an arbitrary extension of L_0 and Θ is a statement

[13] This condition implies that τ is a partial mapping from $\widehat{T}(C)$ to $2^{\widehat{T}(C)\times\widehat{T}(C)}$.

[14] Regularity is the natural formulation of Martin-Löf's semantical criterion for the equality relation in a canonical type.

[15] See Appendix B.

of L, $\mathrm{ITT}_0^L \vdash \Theta$ means that a derivation of Θ can be obtained in ITT_0 by using a valuation of L to get instances of inference rules. The following theorem shows that the system of inference rules given in [14] is built up to meet any series $\{\mathcal{M}^{L_i}\}$ of type systems indexed by a sequence $\{L_i\}_{i\geq 0}$ of extended languages that satisfy both determinism and regularity.

Theorem 5.1 [open-endedness of objects and types] If L is a deterministic and regular extension of L_0, then $\mathrm{ITT}_0^L \vdash \Theta$ implies $\mathcal{M}^L \models \Theta$ for every statement Θ of L.

Proof. Let $L = (C, S)$ be an arbitrary extension of L_0, and suppose that L is deterministic and regular. The theorem is proved by induction on the structure of the derivation of Θ. Consider only the case in which the derivation ends up with an instance

$$\frac{c = f \in \Pi(A, B) \quad a = d \in A}{\mathrm{Ap}(c, a) = \mathrm{Ap}(f, d) \in B(a)}$$

of the hypothesis-free second Π-elimination rule. The treatment of the other cases follows a similar pattern.

By induction hypothesis, we have

$$\begin{array}{ll} \exists\phi.\mathcal{M}^L(\Pi(A,B),\phi) & \& \phi(c,f), \quad \text{and} \\ \exists\psi.\mathcal{M}^L(A,\psi) & \& \psi(a,d). \end{array}$$

Since $\mathcal{I}^L(\mathcal{S}^L, \mathcal{M}^L) = \mathcal{M}^L$, $\mathcal{M}^L(\Pi(A,B),\phi)$ implies $\mathcal{K}^L(\mathcal{M}^L)(\Pi(A,B),\phi)$; so there exist ϕ_A and ϕ_B such that

$$\begin{array}{ll} & \mathcal{M}^L(A,\phi_A) \\ \& \forall u \forall u'.\phi_A(u,u') \Rightarrow & \mathcal{M}^L(B(u),\phi_B(u)) \\ & \& \mathcal{M}^L(B(u'),\phi_B(u')) \\ & \& \forall s \forall s'[\phi_B(u,s,s') \Leftrightarrow \phi_B(u',s,s')] \\ \& \phi = [\![\varphi_0(\Pi)]\!]^L_{A,B,\phi_A,\phi_B}. & \end{array}$$

There also exist b and e such that

$$\begin{array}{l} c \rightarrow_C \lambda(b) \ \& \ f \rightarrow_C \lambda(e) \\ \& \forall u \forall u'.\phi_A(u,u') \Rightarrow \phi_B(u, b(u), e(u')) \end{array}$$

because $[\![\varphi_0(\Pi)]\!]^L_{A,B,\phi_A,\phi_B}(c, f)$. Since $\mathcal{M}^L$ is extensional, $\phi_A = \psi$; so $\phi_A(a, d)$. Hence

$$\mathcal{M}^L(B(a), \phi_B(a)) \ \& \ \phi_B(a, b(a), e(d)).$$

This $b(a)$ can be replaced with $\mathrm{Ap}(c, a)$ because they have the same value and $\mathcal{M}^L$ is extensional. For the same reason, $e(d)$ can be replaced with $\mathrm{Ap}(f, d)$. This proves that $\mathcal{M}^L \models \mathrm{Ap}(c, a) = \mathrm{Ap}(f, d) \in B(a)$. ∎

Corollary 5.2 If L is a deterministic and regular extension of L_0 and if L' is a deterministic and regular extension of L, then $\mathrm{ITT}_0^L \vdash \Theta$ implies $\mathcal{M}^{L'} \models \Theta$ for every statement Θ of L.

Corollary 5.2 is contrasted with the *nonmonotonicity* of a series $\{\mathcal{M}^L\}$ of type systems:

Claim 5.3 It is not true that if L is a deterministic and regular method system and if L' is a deterministic and regular extension of L, then $\mathcal{M}^L \models \Theta$ implies $\mathcal{M}^{L'} \models \Theta$ for every statement Θ of L.

Proof. A counterexample is simply given by taking

Table 7: Method System $L_\nabla = ((O_\nabla, K_\nabla, \alpha_\nabla, R_\nabla), (D_\nabla, \kappa_\nabla, \varphi_\nabla))$.

$$O_\nabla = O_0 \cup \{\mathrm{Urec}_n\,(\text{for every } n \geq 0)\}\,, \quad K_\nabla = K_0,$$

$$\alpha_\nabla(\rho) = \begin{cases} (0,0,0,4,4,4,4,4,\underbrace{0,\ldots,0}_{n}) & \text{if } \rho \text{ is } \mathrm{Urec}_n \text{ for } n \geq 0, \\ \alpha_0(\rho) & \text{otherwise,} \end{cases}$$

$$R_\nabla = R_0 \cup \left\{ \begin{array}{l} \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to \mathrm{N}_k \;\&\; a_1 \to b \\ \quad \text{for every } k \geq 0, \\ \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to \mathrm{N} \;\&\; a_2 \to b, \\ \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to \mathrm{I}(A, c, d) \\ \quad \&\; a_3(A, c, d, \mathrm{Urec}_n(A, a_1, ..., a_{7+n})) \to b, \\ \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to A + B \\ \quad \&\; a_4(A, B, \mathrm{Urec}_n(A, a_1, ..., a_{7+n}), \mathrm{Urec}_n(B, a_1, ..., a_{7+n})) \to b, \\ \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to \Pi(A, B) \\ \quad \&\; a_5(A, \lambda(B), \mathrm{Urec}_n(A, a_1, ..., a_{7+n}), \lambda(w.\mathrm{Urec}_n(B(w), a_1, ..., a_{7+n}))) \to b, \\ \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to \Sigma(A, B) \\ \quad \&\; a_6(A, \lambda(B), \mathrm{Urec}_n(A, a_1, ..., a_{7+n}), \lambda(w.\mathrm{Urec}_n(B(w), a_1, ..., a_{7+n}))) \to b, \\ \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to \mathrm{W}(A, B) \\ \quad \&\; a_7(A, \lambda(B), \mathrm{Urec}_n(A, a_1, ..., a_{7+n}), \lambda(w.\mathrm{Urec}_n(B(w), a_1, ..., a_{7+n}))) \to b, \\ \mathrm{Urec}_n(a, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, ..., a_{7+n}) \to b \Leftarrow a \to \mathrm{U}_k \;\&\; a_{8+k} \to b \\ \quad \text{for every } k \in \{0, 1, \ldots, n-1\}, \\ \text{for every } n \geq 0, \end{array} \right\}$$

$$D_\nabla = D_0, \quad \kappa_\nabla = \kappa_0, \quad \varphi_\nabla = \varphi_0.$$

- as L, the extension L_∇ of L_0 with the "Urec_n" operators for universe elimination, which extension is described in Table 7,
- as L', an arbitrary extension of L_∇ that includes a fresh type constructor and satisfies both determinism and regularity, and
- as Θ, $\Pi(\mathrm{U}_0, v.\mathrm{Urec}_0(v, \mathrm{N}, \mathrm{N}, xyzw.\mathrm{N}, xyzw.\mathrm{N}, xyzw.\mathrm{N}, xyzw.\mathrm{N}, xyzw.\mathrm{N})) \in \mathrm{U}_1$. ■

6 Concluding Remarks

This paper has treated ITT as an open-ended framework essentially consisting of

- flexibly extensible languages,
- their uniform, effectively given semantics, and
- persistently valid inference rules.

This purely mathematical approach may, of course, not deal with a number of philosophical aspects of open-endedness. From the mathematical viewpoint, however, deeper analysis and more general treatment of open-endedness may be based on this formulation. The uniform construction of types can also be used theoretically to prove properties of types other than extensionality "uniformly" instead of "by case analysis." In addition, a prescription for the class of types that can be introduced into the theory—that is, the prescription that this class should consist of types representable in some type system built from a deterministic and regular

extension of the original method system—will be practically useful for checking whether new types under consideration can be introduced.

It will be possible to further extend the class of types that can be validly added. An interesting extension is to allow types that "grow," like universes, through the introduction of their new canonical-object constructors. And, closely related to Corollary 5.2 and Claim 5.3, does the following conjecture—the *conservativeness* of a series $\{\mathcal{M}^L\}$ of type systems—hold?

Conjecture 6.1 If L is a method system and if L' is an extension of L, then $\mathcal{M}^{L'} \models \Theta$ implies $\mathcal{M}^L \models \Theta$ for every statement Θ of L.

Acknowledgments

The author would like to express his sincere gratitude to Dr. Hirofumi Katsuno, Dr. Shigeki Goto, Mr. Mizuhito Ogawa, and Dr. Ronald van der Meyden for their encouragement and helpful advice. The author would also like to thank Professor Masahiko Sato, Mr. Yukiyoshi Kameyama, and Dr. Makoto Tatsuta for their insights, valuable discussions, and helpful comments. Special thanks are also due to Mr. Hiroyuki Shirasu for fruitful discussions on the basic framework underlying this paper. The author also appreciates the excellent comments by Mr. Lachlan Johnson and Dr. Randall Kaul that have considerably improved this paper.

References

[1] Peter Aczel. Frege structures and the notions of proposition, truth and set. In J. Barwise, H. J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, pp. 31–59. North-Holland, 1980.

[2] Peter Aczel, David P. Carlisle, and Nax Mendler. Two frameworks of theories and their implementation in Isabelle. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pp. 3–39. Cambridge University Press, 1991.

[3] Stuart F. Allen. A non-type-theoretic definition of Martin-Löf's types. In *Proceedings of the Second Annual Symposium on Logic in Computer Science*, pp. 215–221. IEEE, 1987.

[4] Stuart F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. PhD thesis, Cornell University, 1987.

[5] Roland Backhouse, Paul Chisholm, and Grant Malcolm. Do-it-yourself type theory (part 1, 2). *BULLETIN of the European Association for Theoretical Computer Science*, No. 34, 35,, 1988.

[6] David A. Basin and Douglas J. Howe. Some normalization properties of Martin-Löf's type theory, and applications. In *Proceedings of the First International Conference on Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pp. 475–494. Springer-Verlag, 1991.

[7] Michael J. Beeson. Recursive models for constructive set theories. *Annals of Mathematical Logic*, Vol. 23, pp. 127–178, 1982.

[8] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.

[9] Thierry Coquand and Christine Paulin. Inductively defined types. In *Proceedings of the International Conference on Computer Logic*, volume 417 of *Lecture Notes in Computer Science*, pp. 50–66. Springer-Verlag, 1988.

[10] Peter Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pp. 280–306. Cambridge University Press, 1991.

[11] Robert Harper. Constructing type systems over an operational semantics. *Journal of Symbolic Computation*, Vol. 14, pp. 71–84, 1992.

[12] Douglas J. Howe. Equality in lazy computation systems. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pp. 198–203. IEEE, 1989.

[13] Douglas J. Howe. On computational open-endedness in Martin-Löf's type theory. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science*, pp. 162–172. IEEE, 1991.

[14] Per Martin-Löf. Constructive mathematics and computer programming. In L. J. Cohen, J. Los, H. Pfeiffer, and K. P. Podewsky, editors, *Logic, Methodology and Philosophy of Science*, pp. 153–175. North-Holland, 1982.

[15] Per Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[16] Per Martin-Löf. Truth of a proposition, evidence of a judgement, validity of a proof. *Synthese*, Vol. 73, pp. 407–420, 1987.

[17] Paul F. Mendler and Peter Aczel. The notion of a framework and a framework for LTC. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pp. 392–399. IEEE, 1988.

[18] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, 1990.

[19] Masahiko Sato. Adding proof objects and inductive definition mechanisms to Frege structures. In *Proceedings of the First International Conference on Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pp. 53–87. Springer-Verlag, 1991.

[20] Jan Smith. An interpretation of Martin-Löf's type theory in a type-free theory of propositions. *Journal of Symbolic Logic*, Vol. 49, pp. 730–753, 1984.

[21] Makoto Tatsuta. Program synthesis using realizability. *Theoretical Computer Science*, Vol. 90, pp. 309–353, 1991.

[22] Makoto Tatsuta. Monotone recursive definition of predicates and its realizability interpretation. In *Proceedings of the First International Conference on Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pp. 38–52. Springer-Verlag, 1991.

[23] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics*, volume I, II. North-Holland, 1988.

A Method System L_0

The original method system $L_0 = (C_0, S_0)$, where $C_0 = (O_0, K_0, \alpha_0, R_0)$ and $S_0 = (D_0, \kappa_0, \varphi_0)$, is described below.

$$O_0 = \left\{ \begin{array}{c} \mathrm{N}_n, m_n, \mathrm{R}_n \text{ (for every } n \geq 0 \text{ and } m \in \{0, 1, \ldots, n-1\}), \\ \mathrm{N}, 0, \mathrm{S}, \mathrm{R}, \mathrm{I}, \mathrm{r}, \mathrm{J}, +, \mathrm{i}, \mathrm{j}, \mathrm{D}, \\ \Pi, \lambda, \mathrm{Ap}, \Sigma, (\cdot, \cdot), \mathrm{E}, \mathrm{W}, \mathrm{sup}, \mathrm{Tr}, \mathrm{U}_n \text{ (for every } n \geq 0) \end{array} \right\}$$

$$K_0 = \left\{ \begin{array}{c} \mathrm{N}_n, m_n \text{ (for every } n \geq 0 \text{ and } m \in \{0, 1, \ldots, n-1\}), \\ \mathrm{N}, 0, \mathrm{S}, \mathrm{I}, \mathrm{r}, +, \mathrm{i}, \mathrm{j}, \Pi, \lambda, \Sigma, (\cdot, \cdot), \mathrm{W}, \mathrm{sup}, \mathrm{U}_n \text{ (for every } n \geq 0) \end{array} \right\}$$

$$\alpha_0(\rho) = \left\{ \begin{array}{ll} () & \text{if } \rho \text{ is } \mathrm{N}_n, m_n, \mathrm{N}, 0, \mathrm{r}, \mathrm{U}_n \text{ for } n \geq 0 \text{ and } m \in \{0, 1, \ldots, n-1\}, \\ (0) & \text{if } \rho \text{ is } \mathrm{S}, \mathrm{i}, \mathrm{j}, \\ (1) & \text{if } \rho \text{ is } \lambda, \\ (0,0) & \text{if } \rho \text{ is } \mathrm{Ap}, (\cdot, \cdot), \mathrm{J}, +, \mathrm{sup}, \\ (0,1) & \text{if } \rho \text{ is } \Pi, \Sigma, \mathrm{W}, \\ (0,2) & \text{if } \rho \text{ is } \mathrm{E}, \\ (0,3) & \text{if } \rho \text{ is } \mathrm{Tr}, \\ (0,0,0) & \text{if } \rho \text{ is } \mathrm{I}, \\ (0,0,2) & \text{if } \rho \text{ is } \mathrm{R}, \\ (0,1,1) & \text{if } \rho \text{ is } \mathrm{D}, \\ (0, \underbrace{0, \ldots, 0}_{n}) & \text{if } \rho \text{ is } \mathrm{R}_n \text{ for } n \geq 0. \end{array} \right.$$

$$R_0 = \left\{ \begin{array}{rcl} \mathrm{R}_n(c, c_0, \ldots, c_{n-1}) \to d & \Leftarrow & c \to m_n \,\&\, c_m \to d \\ & & \text{for every } n \geq 0 \text{ and } m \in \{0, 1, \ldots, n-1\}, \\ \mathrm{R}(c, d, e) \to f & \Leftarrow & c \to 0 \,\&\, d \to f, \\ \mathrm{R}(c, d, e) \to f & \Leftarrow & c \to \mathrm{S}(a) \,\&\, e(a, \mathrm{R}(a, d, e)) \to f, \\ \mathrm{J}(c, d) \to e & \Leftarrow & c \to \mathrm{r} \,\&\, d \to e, \\ \mathrm{D}(c, d, e) \to f & \Leftarrow & c \to \mathrm{i}(a) \,\&\, d(a) \to f, \\ \mathrm{D}(c, d, e) \to f & \Leftarrow & c \to \mathrm{j}(b) \,\&\, e(b) \to f, \\ \mathrm{Ap}(c, a) \to d & \Leftarrow & c \to \lambda(b) \,\&\, b(a) \to d, \\ \mathrm{E}(c, d) \to e & \Leftarrow & c \to (a, b) \,\&\, d(a, b) \to e, \\ \mathrm{Tr}(c, d) \to e & \Leftarrow & c \to \mathrm{sup}(a, b) \,\&\, d(a, b, \lambda(v.\mathrm{Tr}(\mathrm{Ap}(b, v), d))) \to e \end{array} \right\}$$

$$D_0 = \{\mathrm{N}_n \text{ (for every } n \geq 0), \mathrm{N}, \mathrm{I}, +, \Pi, \Sigma, \mathrm{W}\}$$

$$\kappa_0(\Delta) = \begin{cases} \varepsilon & \text{if } \Delta \text{ is } \mathrm{N}_n,\ \mathrm{N}, \\ 1\square\square & \text{if } \Delta \text{ is I}, \\ 11 & \text{if } \Delta \text{ is } +, \\ 12 & \text{if } \Delta \text{ is } \Pi,\ \Sigma,\ \mathrm{W}. \end{cases}$$

$$\varphi_0(\Delta) = \begin{cases} \mu P.(x,x').\ x \to 0_n \ \&\ x' \to 0_n & \text{if } \Delta \text{ is } \mathrm{N}_n, \\ \quad \vee \cdots \vee & \\ \quad x \to (n-1)_n \ \&\ x' \to (n-1)_n & \\ \mu P.(x,x').\ x \to 0 \ \&\ x' \to 0 \ \vee & \text{if } \Delta \text{ is N}, \\ \quad \exists a \exists a'.\ x \to \mathrm{S}(a) \ \&\ x' \to \mathrm{S}(a') \ \&\ P(a,a') & \\ \mu P.(x,x').\ x \to \mathrm{r} \ \&\ x' \to \mathrm{r} \ \&\ Q_{1,1}(F_{1,1,1}, F_{1,1,2}) & \text{if } \Delta \text{ is I}, \\ \mu P.(x,x').\ \exists a \exists a'.\ x \to \mathrm{i}(a) \ \&\ x' \to \mathrm{i}(a') \ \&\ Q_{1,1}(a,a') \ \vee & \text{if } \Delta \text{ is } +, \\ \quad \exists b \exists b'.\ x \to \mathrm{j}(b) \ \&\ x' \to \mathrm{j}(b') \ \&\ Q_{2,1}(b,b') & \\ \mu P.(x,x').\ \exists b \exists b'.\ x \to \lambda(b) \ \&\ x' \to \lambda(b') & \text{if } \Delta \text{ is } \Pi, \\ \quad \&\ \forall v \forall v'.\ Q_{1,1}(v,v') \Rightarrow Q_{1,2}(v, b(v), b'(v')) & \\ \mu P.(x,x').\ \exists a b \exists a' b'.\ x \to (a,b) \ \&\ x' \to (a',b') & \text{if } \Delta \text{ is } \Sigma, \\ \quad \&\ Q_{1,1}(a,a') & \\ \quad \&\ Q_{1,1}(a,a') \Rightarrow Q_{1,2}(a,b,b') & \\ \mu P.(x,x').\ \exists a b \exists a' b'.\ x \to \sup(a,b) \ \&\ x' \to \sup(a',b') & \text{if } \Delta \text{ is W}. \\ \quad \&\ \exists g \exists g'.\ b \to \lambda(g) \ \&\ b' \to \lambda(g') & \\ \quad \&\ Q_{1,1}(a,a') & \\ \quad \&\ Q_{1,1}(a,a') \Rightarrow \forall v \forall v'.\ Q_{1,2}(a,v,v') & \\ \quad \Rightarrow P(g(v), g'(v')) & \end{cases}$$

B Deduction System ITT_0

The system of inference rules given in [14] can be formalized as the deduction system ITT_0, which is built up in the language L_0. The essential part of ITT_0 is described below. A slightly more rigorous description is available in [23], for example.

In this appendix, Latin letters $A, B, C, D, a, b, c, d, \ldots$ denote second-order variables of appropriate arities.

General rules.

Reflexivity:

$$\frac{a \in A}{a = a \in A} \qquad \frac{A \text{ type}}{A = A}$$

Symmetry:

$$\frac{a = b \in A}{b = a \in A} \qquad \frac{A = B}{B = A}$$

Transitivity:

$$\frac{a = b \in A \quad b = c \in A}{a = c \in A} \qquad \frac{A = B \quad B = C}{A = C}$$

Equality of types:

$$\frac{a \in A \quad A = B}{a \in B} \qquad \frac{a = b \in A \quad A = B}{a = b \in B}$$

Substitution:

$$\frac{a \in A \quad B(x)\ \text{type}\ (x \in A)}{B(a)\ \text{type}} \qquad \frac{a = c \in A \quad B(x) = D(x)\ (x \in A)}{B(a) = D(c)}$$

$$\frac{a \in A \quad b(x) \in B(x)\ (x \in A)}{b(a) \in B(a)} \qquad \frac{a = c \in A \quad b(x) = d(x) \in B(x)\ (x \in A)}{b(a) = d(c) \in B(a)}$$

Assumption:

$$\frac{A\ \text{type}}{x \in A\ (x \in A)}$$

Cartesian product of a family of types.

Π-formation:

$$\frac{B(x)\ \text{type}\ (x \in A)}{\Pi(A,B)\ \text{type}} \qquad \frac{A = C \quad B(x) = D(x)\ (x \in A)}{\Pi(A,B) = \Pi(C,D)}$$

Π-introduction:

$$\frac{b(x) \in B(x)\ (x \in A)}{\lambda(b) \in \Pi(A,B)} \qquad \frac{b(x) = d(x) \in B(x)\ (x \in A)}{\lambda(b) = \lambda(d) \in \Pi(A,B)}$$

Π-elimination:

$$\frac{c \in \Pi(A,B) \quad a \in A}{\mathrm{Ap}(c,a) \in B(a)} \qquad \frac{c = f \in \Pi(A,B) \quad a = d \in A}{\mathrm{Ap}(c,a) = \mathrm{Ap}(f,d) \in B(a)}$$

Π-equality:

$$\frac{a \in A \quad b(x) \in B(x)\ (x \in A)}{\mathrm{Ap}(\lambda(b),a) = b(a) \in B(a)} \qquad \frac{c \in \Pi(A,B)}{\lambda(x.\mathrm{Ap}(c,x)) = c \in \Pi(A,B)}$$

Disjoint union of a family of types.

Σ-formation:

$$\frac{B(x)\ \text{type}\ (x \in A)}{\Sigma(A,B)\ \text{type}} \qquad \frac{A = C \quad B(x) = D(x)\ (x \in A)}{\Sigma(A,B) = \Sigma(C,D)}$$

Σ-introduction:

$$\frac{\begin{array}{c}B(x)\ \text{type}\ (x \in A)\\ a \in A \quad b \in B(a)\end{array}}{(a,b) \in \Sigma(A,B)} \qquad \frac{\begin{array}{c}B(x)\ \text{type}\ (x \in A)\\ a = c \in A \quad b = d \in B(a)\end{array}}{(a,b) = (c,d) \in \Sigma(A,B)}$$

Σ-elimination:

$$\frac{\begin{array}{c}C(z)\ \text{type}\ (z \in \Sigma(A,B))\\ c \in \Sigma(A,B) \quad d(x,y) \in C((x,y))\ (x \in A, y \in B(x))\end{array}}{\mathrm{E}(c,d) \in C(c)}$$

$$\frac{\begin{array}{c}C(z)\ \text{type}\ (z \in \Sigma(A,B))\\ c = e \in \Sigma(A,B) \quad d(x,y) = f(x,y) \in C((x,y))\ (x \in A, y \in B(x))\end{array}}{\mathrm{E}(c,d) = \mathrm{E}(e,f) \in C(c)}$$

Σ-equality:

$$\frac{a \in A \quad b \in B(a) \quad d(x,y) \in C((x,y))\ (x \in A, y \in B(x))}{\mathrm{E}((a,b),d) = d(a,b) \in C((a,b))}$$

Disjoint union of two types.

+-formation:

$$\frac{A \text{ type} \quad B \text{ type}}{A + B \text{ type}} \qquad \frac{A = C \quad B = D}{A + B = C + D}$$

+-introduction:

$$\frac{a \in A \quad B \text{ type}}{\mathrm{i}(a) \in A + B} \qquad \frac{a = c \in A \quad B \text{ type}}{\mathrm{i}(a) = \mathrm{i}(c) \in A + B}$$

$$\frac{A \text{ type} \quad b \in B}{\mathrm{j}(b) \in A + B} \qquad \frac{A \text{ type} \quad b = d \in B}{\mathrm{j}(b) = \mathrm{j}(d) \in A + B}$$

+-elimination:

$$\frac{\begin{array}{c} C(z) \text{ type } (z \in A + B) \\ c \in A + B \quad d(x) \in C(\mathrm{i}(x))\ (x \in A) \quad e(y) \in C(\mathrm{j}(y))\ (y \in B) \end{array}}{\mathrm{D}(c, d, e) \in C(c)}$$

$$\frac{\begin{array}{c} C(z) \text{ type } (z \in A + B) \\ c = f \in A + B \quad d(x) = g(x) \in C(\mathrm{i}(x))\ (x \in A) \quad e(y) = h(y) \in C(\mathrm{j}(y))\ (y \in B) \end{array}}{\mathrm{D}(c, d, e) = \mathrm{D}(f, g, h) \in C(c)}$$

+-equality:

$$\frac{a \in A \quad d(x) \in C(\mathrm{i}(x))\ (x \in A) \quad e(y) \in C(\mathrm{j}(y))\ (y \in B)}{\mathrm{D}(\mathrm{i}(a), d, e) = d(a) \in C(\mathrm{i}(a))}$$

$$\frac{b \in B \quad d(x) \in C(\mathrm{i}(x))\ (x \in A) \quad e(y) \in C(\mathrm{j}(y))\ (y \in B)}{\mathrm{D}(\mathrm{j}(b), d, e) = e(b) \in C(\mathrm{j}(b))}$$

Identity relation.

I-formation:

$$\frac{a \in A \quad b \in A}{\mathrm{I}(A, a, b) \text{ type}} \qquad \frac{A = C \quad a = c \in A \quad b = d \in A}{\mathrm{I}(A, a, b) = \mathrm{I}(C, c, d)}$$

I-introduction:

$$\frac{a = b \in A}{\mathrm{r} \in \mathrm{I}(A, a, b)} \qquad \frac{a = b \in A}{\mathrm{r} = \mathrm{r} \in \mathrm{I}(A, a, b)}$$

I-elimination:

$$\frac{c \in \mathrm{I}(A, a, b)}{a = b \in A}$$

$$\frac{C(z) \text{ type } (z \in \mathrm{I}(A, a, b)) \quad c \in \mathrm{I}(A, a, b) \quad d \in C(\mathrm{r})}{\mathrm{J}(c, d) \in C(c)}$$

$$\frac{C(z) \text{ type } (z \in \mathrm{I}(A, a, b)) \quad c = e \in \mathrm{I}(A, a, b) \quad d = f \in C(\mathrm{r})}{\mathrm{J}(c, d) = \mathrm{J}(e, f) \in C(c)}$$

I-equality:

$$\frac{a = b \in A \quad d \in C(\mathrm{r})}{\mathrm{J}(\mathrm{r}, d) = d \in C(\mathrm{r})}$$

Finite types.

N_n-formation:

$$\mathrm{N}_n \text{ type} \quad \mathrm{N}_n = \mathrm{N}_n \quad (n = 0, 1, \ldots)$$

N_n-introduction:

$$m_n \in \mathrm{N}_n \quad m_n = m_n \in \mathrm{N}_n \quad (m = 0, 1, \ldots, n-1)$$

N_n-elimination:

$$\frac{C(z)\ \text{type}\ (z \in \mathrm{N}_n) \quad c \in \mathrm{N}_n \quad c_m \in C(m_n) \quad (m = 0, 1, \ldots, n-1)}{\mathrm{R}_n(c, c_0, \ldots, c_{n-1}) \in C(c)}$$

$$\frac{C(z)\ \text{type}\ (z \in \mathrm{N}_n) \quad c = d \in \mathrm{N}_n \quad c_m = d_m \in C(m_n) \quad (m = 0, 1, \ldots, n-1)}{\mathrm{R}_n(c, c_0, \ldots, c_{n-1}) = \mathrm{R}_n(d, d_0, \ldots, d_{n-1}) \in C(c)}$$

N_n-equality:

$$\frac{c_m \in C(m_n)}{\mathrm{R}_n(m_n, c_0, \ldots, c_{n-1}) = c_m \in C(m_n)} \quad (m = 0, 1, \ldots, n-1)$$

Natural numbers.

N-formation:

$$\mathrm{N}\ \text{type} \qquad \mathrm{N} = \mathrm{N}$$

N-introduction:

$$0 \in \mathrm{N} \quad \frac{a \in \mathrm{N}}{\mathrm{S}(a) \in \mathrm{N}} \quad 0 = 0 \in \mathrm{N} \quad \frac{a = b \in \mathrm{N}}{\mathrm{S}(a) = \mathrm{S}(b) \in \mathrm{N}}$$

N-elimination:

$$\frac{c \in \mathrm{N} \quad d \in C(0) \quad e(x, y) \in C(\mathrm{S}(x))\ (x \in \mathrm{N}, y \in C(x))}{\mathrm{R}(c, d, e) \in C(c)}$$

$$\frac{c = f \in \mathrm{N} \quad d = g \in C(0) \quad e(x, y) = h(x, y) \in C(\mathrm{S}(x))\ (x \in \mathrm{N}, y \in C(x))}{\mathrm{R}(c, d, e) = \mathrm{R}(f, g, h) \in C(c)}$$

N-equality:

$$\frac{d \in C(0) \quad e(x, y) \in C(\mathrm{S}(x))\ (x \in \mathrm{N}, y \in C(x))}{\mathrm{R}(0, d, e) \in C(0)}$$

$$\frac{a \in \mathrm{N} \quad d \in C(0) \quad e(x, y) \in C(\mathrm{S}(x))\ (x \in \mathrm{N}, y \in C(x))}{\mathrm{R}(\mathrm{S}(a), d, e) = e(a, \mathrm{R}(a, d, e)) \in C(\mathrm{S}(a))}$$

Well-orderings.

W-formation:

$$\frac{B(x)\ \text{type}\ (x \in A)}{\mathrm{W}(A, B)\ \text{type}} \quad \frac{A = C \quad B(x) = D(x)\ (x \in A)}{\mathrm{W}(A, B) = \mathrm{W}(C, D)}$$

W-introduction:

$$\frac{a \in A \quad b \in B(a) \to \mathrm{W}(A, B)}{\sup(a, b) \in \mathrm{W}(A, B)} \quad \frac{a = c \in A \quad b = d \in B(a) \to \mathrm{W}(A, B)}{\sup(a, b) = \sup(c, d) \in \mathrm{W}(A, B)}$$

W-elimination:

$$\frac{\begin{array}{c} C(w)\ \text{type}\ (w \in \mathrm{W}(A, B)) \\ c \in \mathrm{W}(A, B) \\ d(x, y, z) \in C(\sup(x, y))\ (x \in A, y \in B(x) \to \mathrm{W}(A, B), z \in \Pi(B(x), v.C(\mathrm{Ap}(y, v)))) \end{array}}{\mathrm{Tr}(c, d) \in C(c)}$$

$$\frac{\begin{array}{c} C(w)\ \text{type}\ (w \in \mathrm{W}(A,B)) \\ c = e \in \mathrm{W}(A,B) \\ d(x,y,z) = f(x,y,z) \in C(\mathrm{sup}(x,y))\ (x \in A, y \in B(x) \to \mathrm{W}(A,B), z \in \Pi(B(x), v.C(\mathrm{Ap}(y,v)))) \end{array}}{\mathrm{Tr}(c,d) = \mathrm{Tr}(e,f) \in C(c)}$$

W-equality:

$$\frac{\begin{array}{c} a \in A \\ b \in B(a) \to \mathrm{W}(A,B) \\ d(x,y,z) \in C(\mathrm{sup}(x,y))\ (x \in A, y \in B(x) \to \mathrm{W}(A,B), z \in \Pi(B(x), v.C(\mathrm{Ap}(y,v)))) \end{array}}{\mathrm{Tr}(\mathrm{sup}(a,b),d) = d(a,b,\lambda(v.\mathrm{Tr}(\mathrm{Ap}(b,v),d))) \in C(\mathrm{sup}(a,b))}$$

Universes.

U_n-formation:

$$\mathrm{U}_n\ \text{type}\quad \mathrm{U}_n = \mathrm{U}_n$$

U_n-introduction:

$$\frac{A \in \mathrm{U}_n \quad B(x) \in \mathrm{U}_n\ (x \in A)}{\Pi(A,B) \in \mathrm{U}_n} \qquad \frac{A = C \in \mathrm{U}_n \quad B(x) = D(x) \in \mathrm{U}_n\ (x \in A)}{\Pi(A,B) = \Pi(C,D) \in \mathrm{U}_n}$$

$$\frac{A \in \mathrm{U}_n \quad B(x) \in \mathrm{U}_n\ (x \in A)}{\Sigma(A,B) \in \mathrm{U}_n} \qquad \frac{A = C \in \mathrm{U}_n \quad B(x) = D(x) \in \mathrm{U}_n\ (x \in A)}{\Sigma(A,B) = \Sigma(C,D) \in \mathrm{U}_n}$$

$$\frac{A \in \mathrm{U}_n \quad B \in \mathrm{U}_n}{A + B \in \mathrm{U}_n} \qquad \frac{A = C \in \mathrm{U}_n \quad B = D \in \mathrm{U}_n}{A + B = C + D \in \mathrm{U}_n}$$

$$\frac{A \in \mathrm{U}_n \quad a \in A \quad b \in A}{\mathrm{I}(A,a,b) \in \mathrm{U}_n} \qquad \frac{A = C \in \mathrm{U}_n \quad a = c \in A \quad b = d \in A}{\mathrm{I}(A,a,b) = \mathrm{I}(C,c,d) \in \mathrm{U}_n}$$

$$\begin{array}{ll} \mathrm{N}_0 \in \mathrm{U}_n & \mathrm{N}_0 = \mathrm{N}_0 \in \mathrm{U}_n \\ \mathrm{N}_1 \in \mathrm{U}_n & \mathrm{N}_1 = \mathrm{N}_1 \in \mathrm{U}_n \\ \vdots & \vdots \\ \mathrm{N} \in \mathrm{U}_n & \mathrm{N} = \mathrm{N} \in \mathrm{U}_n \end{array}$$

$$\frac{A \in \mathrm{U}_n \quad B(x) \in \mathrm{U}_n\ (x \in A)}{\mathrm{W}(A,B) \in \mathrm{U}_n} \qquad \frac{A = C \in \mathrm{U}_n \quad B(x) = D(x) \in \mathrm{U}_n\ (x \in A)}{\mathrm{W}(A,B) = \mathrm{W}(C,D) \in \mathrm{U}_n}$$

$$\begin{array}{ll} \mathrm{U}_0 \in \mathrm{U}_n & \mathrm{U}_0 = \mathrm{U}_0 \in \mathrm{U}_n \\ \vdots & \vdots \\ \mathrm{U}_{n-1} \in \mathrm{U}_n & \mathrm{U}_{n-1} = \mathrm{U}_{n-1} \in \mathrm{U}_n \end{array}$$

U_n-elimination:

$$\frac{A \in \mathrm{U}_n}{A\ \text{type}} \qquad \frac{A = B \in \mathrm{U}_n}{A = B}$$

$$\frac{A \in \mathrm{U}_n}{A \in \mathrm{U}_{n+1}} \qquad \frac{A = B \in \mathrm{U}_n}{A = B \in \mathrm{U}_{n+1}}$$